IEEE488 Interface Expansion Card

for Acorn RISC OS-based Computers

User Guide

Intelligent Interfaces
November 1995

C Copyright Intelligent Interfaces Ltd 1988

Neither the whole or any part of the information contained in this User Guide may be adapted or reproduced in any material form except with the written approval of Intelligent Interfaces Ltd.

All information is given by Intelligent Interfaces in good faith. However, it is acknowledged that there may be errors or omissions in this User Guide. Intelligent Interfaces welcomes comments and suggestions relating to the IEEE488 Interface Expansion Card and this User Guide.

All correspondence should be addressed to:-

Technical Enquiries
Intelligent Interfaces Ltd
P O Box 80
Eastleigh
Hampshire
SO53 2YX

Tel: 023 8026 1514
Fax: 087 0052 1281

This User Guide is intended only to assist the reader in the use of the IEEE488 Interface Expansion Card and, therefore, Intelligent Interfaces shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this User Guide, or any incorrect use of the interface.

CONTENTS

1 Introduction

1.1 Setting the IEEE488 Address of the Computer
1.2 Fitting the Expansion Card
1.3 Connecting Devices to the Computer
1.4 The Scope of the User Guide

2 The IEEE488 Standard

2.1 IEEE488 Device Addresses
2.2 Types of IEEE488 Devices
2.2.1 The Computer as System Controller
2.2.2 The Computer as a Simple Talker/Listener

3 The IEEE488 Interface Software

3.1 Features of the Interface Software
3.1.1 Error Status
3.1.2 Output Modes
3.1.3 Input Modes
3.1.4 End of String Characters for Output
3.1.5 End of String Characters for Input
3.1.6 Timeouts

4 The Computer as Controller in Charge

4.1 Data Output and Input - strings of up to 255 characters
4.1.1 Device with Primary Address Only
4.1.2 Device with Primary and Secondary Address
4.2 Converting between String and Numeric Variables
4.3 Data Output and Input - directly from and to memory
4.3.1 Device with Primary Address Only
4.3.2 Device with Primary and Secondary Address
4.4 Data Transfer between Device
4.5 Determining the Status of a Device
4.5.1 Serial Poll
4.5.2 Parallel Poll
4.6 Initialising Devices
4.7 Triggering Devices
4.8 Remote and Local Operation of Device
4.9 Passing Control to Another Device

5 The Computer as a Simple Talker/Listener

5.1 Data Input and Output - strings of up to 255 characters
5.2 Data Input and Output - directly to and from memory
5.3 Requesting Service from the Controller in Charge

6 BASIC Procedures and Functions

7 Using the IEEE488 Interface from Assembler

8 Hardware

References

Appendices

I Expansion Card DIP Switch Settings
II IEEE488 Interface Commands
III Parallel Poll Response Byte
IV IEEE488 Errors
V Interface Specification
VI BASIC Procedures and Functions

1 Introduction

Before using the IEEE488 Interface Expansion Card check that the following items in addition to this User Guide have been received

IEEE488 Interface Expansion Card
Software Distribution Disc

If any item is missing contact the supplier.

1.1 Setting the IEEE488 Address of the Computer

Switches 1 to 5 of the DIP switch on the IEEE488 Interface Expansion Card specify the IEEE488 Interface address of the Computer. Switch 6 of the DIP switch specifies whether the Computer is either the system controller (off) or a simple talker/listener (on). The Expansion Card is supplied with all switches in the off position ie the address of the Computer is 0 and it is the system controller. Normally, it is not necessary to change these settings. However, if another device has an address of 0 or it is required to use the Computer as a simple talker/listener refer to Appendix I which lists the DIP switch settings.

1.2 Fitting the Expansion Card

The card can be fitted in any Acorn Computer with an expansion backplane. The card can be fitted in the external expansion card socket of an A3000 computer.

To fit the card in an A300 series, A400 series, A540, A5000, Risc PC or A7000 computer:-

1 Switch off the power to the Computer.
2 Disconnect the Computer from the mains supply.
3 The card can be fitted in any unused expansion card slot.
4 Remove the blanking plate from the rear of the Computer and retain the two screws.
5 Fit the card and secure it in position using the two screws retained at stage 4 (if required, fit a joiner and blanking plate).
6 Reconnect the Computer to the mains supply.
7 Switch on the power to the Computer.
8 Confirm that the card has been fitted correctly by typing

```
*PODULES
```

This should list the IEEE488 Interface as

```
Intelligent Interfaces IEEE488 Interface
```

together with any other cards fitted.

Only one IEEE488 Interface Expansion Card can be fitted.

1.3 Connecting Devices to the Computer

Before connecting devices to the Computer ensure that the power to both the Computer and the devices is switched off.

The Computer can be connected to up to 14 other devices using IEEE488 standard cable assemblies. These have a plug and socket at each end, permitting star or linear interconnection of devices. The connectors are provided with two securing screws which allow them to be stacked on the socket of each device. The IEEE488 socket is normally found on the rear of a device. The IEEE488 standard permits individual cable lengths of up to 4m and a total cable length in a system of 2m per device or 20m whichever is the shorter.

When all devices have been connected to the Computer switch on the power to the devices and then the Computer.

1.4 The Scope of the User Guide

This User Guide does not attempt to reproduce the IEEE 488 Standard or explain every aspect of it. The interested user is referred to Ref 1. However, this User Guide provides all the information required to use the Acorn Archimedes Computer fitted with an Intelligent Interfaces IEEE488 Interface Expansion Card with any other IEEE488 device when programming in BBC BASIC or ARM Assembler. Separate libraries of procedures are available from Intelligent Interfaces to simplify the use of the interface from FORTRAN 77 and ANSI C. Details are given in the User Guide supplied with each library.

2 The IEEE488 Standard

The IEEE488 Interface is a general purpose system for exchanging digital data in bit parallel, byte serial form between a number of devices eg digital voltmeters, digital oscilloscopes, spectrum analysers, transient data recorders, function generators etc., in a local area. The original concept was developed by Hewlett Packard who called it the HP-IB (Hewlett Packard Interface Bus). The IEEE (Institute of Electrical and Electronics Engineers) first produced a standard in 1975, IEEE Std 488-1975. This was revised in 1978 as IEEE Std 488-1978. A supplement was included in 1980, IEEE Std 488A-1980. In 1987 the IEEE re-designated IEEE Std 488 as IEEE Std 488.1 and published a draft standard IEEE Std 488.2 'Codes, Formats, Protocols and Common Commands' for use with IEEE Std 488.1.The interface is identical,apart from the mechanical connection system, to that described in the IEC (International Electrotechnical Commission) Publication IEC 625-1, and BSI (British Standards Institute) Publication BS6146 Part 1:1981 'An interface system for programmable measuring instruments (byte-serial, bit-parallel)'. The IEEE Standard specifies a 24 pin Micro-ribbon connector whereas the IEC Publication specifies a 25 pin D-type connector as used for the EIA RS232-C data communication interface.

2.1 IEEE488 Device Addresses

Up to 15 devices can be connected together to form an IEEE488 system. Each device must have a unique address. Some devices have only one address (a primary address) whilst others have an extended address (a primary and a secondary address). Primary and secondary addresses may be in the range 0 to 30. Secondary addresses are often used to select different functions within the same device eg an analogue to digital converter with a number of analogues inputs may have the input selected by the secondary address. Most devices are delivered with factory set primary and secondary addresses. These can be found in the manufacturer's manual. If a number of devices in a system have the same address, then these must be changed by following the manufacturer's instructions. Usually this involves changing DIP switch settings on the rear of the device.

2.2 Types of IEEE488 Devices

A device can have the ability to send data (act as a talker), receive data (act as a listener) or do both (act as a talker/ listener). Some devices also have the ability to manage the system (act as a controller in charge) although there can only be one device acting as the controller in charge at any time. Only one device in an IEEE488 system can act as the system controller. This device and only this device has the ability to send REN (remote enable) and IFC (interface clear) and then operate as the first controller in charge. The Computer is normally the system controller.

2.2.1 The Computer as System Controller

The Computer can be configured as the system controller by setting DIP Switch 6 on the Expansion Card to the off position. As the system controller the Computer can send REN (remote enable) and IFC (interface clear) and then operate as the controller in charge. As controller in charge the Computer manages the interface. It can address itself and other devices to talk and listen and then input and output data. It can send all addressed commands eg GTL (go to local) and all universal commands eg LLO (local lockout). It can conduct both serial and parallel polls and pass control to another device.

2.2.2 The Computer as a Simple Talker/Listener

The Computer can be configured as a simple talker/listener by setting DIP Switch 6 on the Expansion Card to the on position. As a talker/listener the Computer can, when addressed by the controller in charge, input and output data. It can specify the serial poll STB (status byte) and RQS (request service). It can take part in a serial poll and be remotely configured to take part in a parallel poll. It can become controller in charge if control is passed to it by the current controller in charge.

3 The IEEE488 Interface Software

The interface software is a relocatable module which extends the operating system to enable IEEE488 interface operation. The module is stored in ROM on the Expansion Card and is automatically loaded and initialised by the operating system when the Computer is switched on or reset by pressing <Ctrl> <RESET>. If the Computer is configured as the system controller it sends REN (remote enable) true and IFC (interface clear) true for approximately 650 microseconds and becomes the controller in charge. To check that the module has been loaded type

```
*MODULES
```

This lists all the modules present. Once initialised the interface software remains active until the module is either deleted or re-initialised. To delete the module type

```
*RMKILL IEEE488
```

The extensive facilities of the IEEE488 interface software can be accessed through the single Intelligent Interfaces IEEE488 SWI (SoftWare Interrupt) instruction &40340 allocated by Acorn Computers.

To simplify the use of the interface from BASIC two libraries of BASIC procedures and functions are supplied on the software distribution disc. Those in the file BAS488 in the directory BASICProcs abort on error, whereas those in the file XBAS488 in the directory BASICProcs return an error status. The procedures and functions can be INSTALLed in a BASIC library.

```
>INSTALL"$.BASICProcs.BAS488"
```

or

```
>INSTALL"$.BASICProcs.XBAS488"
```

3.1 Features of the Interface Software

3.1.1 Error Status

The functions in the file XBAS488 which return an error status can be used to trap errors within a program and then take the appropriate action. This makes it possible to recover from an error when it occurs within a procedure, function or loop. The error status returned is an IEEE488 Interface Software basic error number. These are listed in Appendix IV.

Some functions, particularly those not associated with bus activity, always return an error status of 0 indicating that no IEEE488 error has occurred. Therefore, for these functions, the return error status form does not offer any advantage over the abort on error form.

3.1.2 Output Modes

When the interface software outputs data it can do so in one of four modes.

Mode 0 - output data and append end of string character or characters.
Mode 1 - output data and send EOI (end or identify) true with the last data byte.
Mode 2 - output n data bytes.
Mode 3 - output data and append end of string character or characters and set EOI true with the last end of string character.

3.1.3 Input Modes

When the interface software inputs data it can do so in one of four modes.

Mode 0 - input data until end of string character or characters are received
Mode 1 - input data until EOI (end or identify) true is received with a data byte.

Mode 2 - input n data bytes.
Mode 3 - input data until end of string character or characters are received and/or until EOI true is received with a data byte.

The most commonly used mode for both the output and input of data is mode 0. This mode is often used by instruments manufactured by Hewlett Packard, Keithley, Fluke etc.

Mode 2 is useful when sending or receiving binary coded data, particularly directly from and to the memory of the Computer eg when writing to or reading from the memory buffer of a spectrum analyser.

Mode 1 is used when a device sends or receives EOI (end or identify) true with the last data byte and is often an alternative to mode 2.

Mode 3 was used by the Commodore Business Computers (CBM) ' PET' Computer and may be required when using devices originally designed for that Computer.

3.1.4 End of String Characters for Output

When data is sent to a device delimiting characters may be required. Consult the manufacturer' s manual to determine the delimiting character or characters that the device requires. When data is output in modes 0 or 3, one or two end of string characters are appended. The interface software defaults to two end of string characters, CR (carriage return, ASCII code 13) and LF (line feed, ASCII code 10). To redefine the end of string characters as LF, CR

```
PROCeos_out_2chr(10,13)
```

or

```
status% = FNXeos_out_2chr(10,13)
```

or to define a single end of string character as LF

```
PROCeos_out_1chr(10)
```

or

```
status% = FNXeos_out_1chr(10)
```

3.1.5 End of String Characters for Input

When data is received from a device delimiting characters may be sent. Consult the manufacturer' s manual to determine the delimiting character or characters that the device sends. In modes 0 or 3 data is input until one or two end of string characters are received. The interface software defaults to two end of string characters, CR (carriage return, ASCII code 13) and LF (line feed, ASCII code 10). To redefine the end of string characters as LF, CR

```
PROCeos_in_2chr(10,13)
```

or

```
status% = FNXeos_in_2chr(10,13)
```

or to define a single end of string character as LF

```
PROCeos_in_1chr(10)
```

or

```
status% = FNXeos_in_1chr(10)
```

3.1.6 Timeouts

As timeouts are not part of the IEEE488 standard the interface software defaults to timeout off. However, it is recommended, particularly during program development, that timeouts are used. To enable timeouts

```
PROCtimeout_on
```

or

```
status% = FNXtimeout_on
```

The default timeout period is approximately eight seconds. To redefine the timeout period as 20 seconds

```
PROCtimeout_period(20)
```

or

```
status% = FNXtimeout_period(20)
```

To disable timeouts

```
PROCtimeout_off
```

or

```
status% = FNXtimeout_off
```

4 The Computer as Controller in Charge

The Computer acts as the controller in charge if

Switch 6 of the DIP switch on the Expansion Card is in the off position ie it is the system controller and the Computer has sent IFC (interface clear)

or

as a simple talker/listener the current controller in charge passes control back to it. This process is transparent to the user and can occur at any time.

To determine whether the Computer is controller in charge

```
state% = FNcontroller_in_charge
```

or

```
status% = FNXcontroller_in_charge(state%)
```

returns the logical state TRUE in the variable state% if the Computer is the controller in charge.

4.1 Data Output and Input - strings of up to 255 characters

4.1.1 Device with Primary Address Only

Example
To abort on error

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM strings of up to 255 characters
1030 REM abort on error
1040
1050 PROCeos_out_1chr(13)
1060 PROCeos_in_1chr(13)
1070 PROCtimeout_on
1080 PROCinit
1090 PROCadrandoutput(0,16,"F0T1R3X")
1100 data$ = FNadrandinput(0,16,255)
  .
  .
```

or to return error status

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM strings of up to 255 characters
1030 REM return error status
1040
1050 status% = FNXeos_out_1chr(13)
1055 IF status% <> 0 THEN 10000
1060 status% = FNXeos_in_1chr(13)
1065 IF status% <> 0 THEN 10000
1070 status% = FNXtimeout_on
1075 IF status% <> 0 THEN 10000
1080 status% = FNXinit
1085 IF status% <> 0 THEN 10000
1090 status% = FNXadrandoutput(0,16,"F0T1R3X")
1095 IF status% <> 0 THEN 10000
1100 status% = FNXadrandinput(data$,0,16,255)
```

```
1105 IF status% <> 0 THEN 10000
 .
 .
10000 IF status% =  1 THEN PRINT "Not initialised"
10010 IF status% =  2 THEN PRINT "Not sysctrl"
10020 IF status% =  3 THEN PRINT "Not ctrlinchg"
10030 IF status% =  4 THEN PRINT "Ctrlinchg"
10040 IF status% =  5 THEN PRINT "Interface clear"
10050 IF status% =  6 THEN PRINT "Sherror"
10060 IF status% =  7 THEN PRINT "Timeout"
10070 IF status% =  8 THEN PRINT "Byte count = 0"
10080 IF status% =  9 THEN PRINT "Sysctrl"
10090 IF status% = 10 THEN PRINT "Buffer full"
10100 IF status% = 11 THEN PRINT "Buffer overflow"
10110 IF status% = 12 THEN PRINT "Unrecognised SWI"
10120 IF status% = 13 THEN PRINT "Unrecognised command"
 .
 .
```

Line 1050 specifies the end of string character for output as CR (carriage return, ASCII code 13) which is required by the instrument

Line 1060 specifies the end of string character for input as CR (carriage return, ASCII code 13) which is sent by the instrument

It is only necessary to define the end of string characters if they differ from the default CR (carriage return), LF (line feed).

Line 1070 enable timeouts

Timeouts are particularly useful during program development.

Line 1080 initialises the interface hardware, sends REN (remote enable) true and IFC (interface clear) true for approximately 650 microseconds. The Computer becomes controller in charge

Line 1090 addresses device 16 to listen and outputs the string F0T1R3X in mode 0 ie the end of string character specified on line 1050 is appended

Line 1100 addresses device 16 to talk and inputs up to 255 characters in mode 0 ie until the end of string character specified on line 1060 is received and returns it in the variable data$

If an error occurs both of the above examples produce a similar result ie an error message is displayed and the program terminates. However, in practice the functions that return error status can be used to trap errors within the program and take the appropriate corrective action.

4.1.2 Device with Primary and Secondary Addresses

Example
To abort on error

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM strings of up to 255 characters
1030 REM abort on error
1040
1050 PROCeos_out_1chr(13)
1060 PROCeos_in_1chr(13)
1070 PROCtimeout_on
1080 PROCinit
1090 PROCadrextandoutput(0,6,3,"F0T1R3X")
1100 data$ = FNadrextandinput(0,6,3,255)
```

```
 .
 .
```

or to return error status

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM strings of up to 255 characters
1030 REM return error status
1040
1050 status% = FNXeos_out_1chr(13)
1055 IF status% <> 0 THEN 10000
1060 status% = FNXeos_in_1chr(13)
1065 IF status% <> 0 THEN 10000
1070 status% = FNXtimeout_on
1075 IF status% <> 0 THEN 10000
1080 status% = FNXinit
1085 IF status% <> 0 THEN 10000
1090 status% = FNXadrextandoutput(0,6,3,"F0T1R3X")
1095 IF status% <> 0 THEN 10000
1100 status% = FNXadrextandinput(data$,0,6,3,255)
1105 IF status% <> 0 THEN 10000
 .
 .
```

Line 1090 addresses device with primary address 6 and secondary address 3 to listen and outputs the string F0T1R3X in mode 0 ie the end of string character specified on line 1050 is appended

Line 1100 addresses device with primary address 6 and secondary address 3 to talk and inputs data in mode 0 ie until the end of string character specified on line 1060 is received and returns it in the variable data$

4.2 Converting between String and Numeric Variables

In order to use data that has been input as a string of ASCII characters in subsequent calculations it must be converted from a string to a real or integer variable. To convert from a string to a real variable

```
reading = VAL (reading$)
```

If the string has a prefix or suffix then this must be stripped off prior to conversion. To convert from a string of format V±n.nE±nn to a real variable

```
reading = VAL(RIGHT$(reading$,LEN(reading$)-1))
```

To convert from a real variable to a string variable

```
value$ = STR$(value)
```

4.3 Data Output and Input - directly from and to memory

Some devices such as spectrum analysers, digital storage oscilloscopes and other microprocessor-based instruments with their own memory can receive and send long fixed length sequences of binary coded data. These are often longer than the maximum 255 characters that can be stored in a string and, therefore, modes 1 or 2 are normally used when outputting or inputting data.

4.3.1 Device with Primary Address Only

The following example is part of a program that was written for a Hewlett Packard HP3582A Spectrum Analyzer in order to read the display of the analyser, save it in a file on disc, and, at a later stage, load it from the file on disc and write it to the display of the analyser.

Example
To abort on error

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM binary coded data directly from and to memory
1030 REM abort on error
1040
1050 DIM buffer% 1025
1060 displaystate% = buffer%
1070 display% = buffer%+2
1080 PROCtimeout_on
1090 PROCinit
1100 PROCadrandoutput(0,11,"HLTLFM,77455,1")
1110 PROCadrandinputmem(2,11,displaystate%,2)
1120 PROCadrandoutput(0,11,"LFM,74000,512")
1130 PROCadrandinputmem(2,11,display%,1024)
1140
1150 REM save data in file on disc
1160
1170 OSCLI "SAVE display "+STR$~buffer%+" + 402"
 .
 .
2960 REM load data from file on disc
2970
2980 OSCLI "LOAD display "+STR$~buffer%
2990
3000 PROCadrandoutput(0,11,"HLTWTM,77455,1")
3010 PROCadrandoutputmem(2,11,displaystate%,2")
3020 PROCadrandoutput(0,11,"WTM,74000,512")
3030 PROCadrandoutputmem(2,11,display%,1024)
 .
 .
```

To return error status

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM binary coded data directly from and to memory
1030 REM return error status
1040
1050 DIM buffer% 1025
1060 displaystate% = buffer%
1070 display% = buffer%+2
1080 status% = FNXtimeout_on
1085 IF status% <> 0 THEN 10000
1090 status% = FNXinit
1095 IF status% <> 0 THEN 10000
1100 status% = FNXadrandoutput(0,11,"HLTLFM,77455,1")
1105 IF status% <> 0 THEN 10000
1110 status% = FNXadrandinputmem(2,11,displaystate%,2)
1115 IF status% <> 0 THEN 10000
1120 status% = FNXadrandoutput(0,11,"LFM,74000,512")
1125 IF status% <> 0 THEN 10000
1130 status% = FNXadrandinputmem(2,11,display%,1024)
1135 IF status% <> 0 THEN 10000
1140
1150 REM save data in file on disc
1160
1170 OSCLI "SAVE display "+STR$~buffer%+" + 402"
```

```
 .
 .
2960 REM load data from file on disc
2970
2980 OSCLI "LOAD display "+STR$~buffer%
2990
3000 status% = FNXadrandoutput(0,11,"HLTWTM,77455,1")
3005 IF status% <> 0 THEN 10000
3010 status% = FNXadrandoutputmem(2,11,displaystate%,2")
3015 IF status% <> 0 THEN 10000
3020 status% = FNXadrandoutput(0,11,"WTM,74000,512")
3025 IF status% <> 0 THEN 10000
3030 status% = FNXadrandoutputmem(2,11,display%,1024)
3035 IF status% <> 0 THEN 10000
 .
 .
```

Line 1050 reserves 1026 bytes of memory, 2 bytes for the displaystate and 1024 byte for the display

Line 1100 addresses device 11, the analyser, to listen and outputs the string HLTLFM,77455,1 in mode 0 ie the default end of string characters CR (carriage return) and LF (line feed) are appended. The string halts (HLT) the analyser and prepares it to list from its memory (LFM) the 16-bit switch register word that describes its display state.

Line 1110 addresses device 11, the analyser, to talk and inputs two bytes of data in mode 2 and stores it in memory starting at address displaystate%

Line 1120 addresses device 11, the analyser, to listen and outputs the string LFM,74000,512 in mode 0. The string prepares the analyser to list from its memory (LFM) the 512 16-bit words of its display buffer.

Line 1130 addresses device 11, the analyser, to talk and inputs 1024 bytes of data in mode 2 and stores it in memory starting at address display%

Line 3000 addresses device 11, the analyser, to listen and outputs the string HLTWTM,77455,1 in mode 0. The string halts (HLT) the analyser and prepares to write to its memory (WTM) the 16-bit switch register word that describes its display state.

Line 3010 addresses device 11, the analyser, to listen and outputs the two bytes of data from memory starting at address displaystate% in mode 2.

Line 3020 addresses device 11, the analyser, to listen and outputs the string WTM,74000,512 in mode 0. The string prepares the analyser to write to its memory (WTM) the 512 16-bit words.

Line 3030 addresses device 11, the analyser, to listen and outputs 1024 bytes of data in mode 2 and stores it in the display buffer of the analyser.

4.3.2 Device with Primary and Secondary Address

Example
To abort on error

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM binary coded data directly from and to memory
1030 REM abort on error
1040
1050 DIM buffer% 511
1060 PROCtimeout_on
1070 PROCinit
1080 PROCadrextandinputmem(2,9,1,buffer%,512)
```

```
1090
1100 REM save data in file on disc
1110
1120 OSCLI "SAVE trace "+STR$~buffer%+" + 200"
 .
 .
2960 REM load data from file on disc
2970
2980 OSCLI "LOAD trace "+STR$~buffer%
2990
3000 PROCadrextandoutputmem(1,9,1,buffer%,512)
 .
 .
```

To return error status

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM binary coded data directly from and to memory
1030 REM return error status
1040
1050 DIM buffer% 511
1060 status% = FNXtimeout_on
1065 IF status% <> 0 THEN 10000
1070 status% = FNXinit
1075 IF status% <> 0 THEN 10000
1080 status% = FNXadrextandinputmem(2,9,1,buffer%,512)
1085 IF status% <> 0 THEN 10000
1090
1100 REM save data in file on disc
1110
1120 OSCLI "SAVE trace "+STR$~buffer%+" + 200"
 .
 .
2960 REM load data from file on disc
2970
2980 OSCLI "LOAD trace "+STR$~buffer%
2990
3000 status% = FNXadrextandoutputmem(1,9,1,buffer%,512)
3005 IF status% <> 0 THEN 10000
 .
 .
```

Line 1050 reserves 512 bytes of memory for the trace.

Line 1080 addresses device with primary address 9 and secondary address 1, the oscilloscope, to talk and inputs 512 bytes of data in mode 2 and stores it in memory starting it at address buffer%.

Line 3000 addresses device with primary address 9 and secondary address 1, the oscilloscope, to listen and outputs 512 bytes of data in mode 1 ie EOI (end or identify) is set true with the last data byte.

4.4 Data Transfer between Devices

If it is required to transfer data between devices (one talker and one or more listeners) without involving the Computer the following sequence should be used.

Example
To abort on error

```
1000 REM computer configured as system controller
1010 REM data transfer
```

```
1020 REM abort on error
1030
1040 PROCinit
1050 PROCunlisten
1060 PROCtalk(7)
1070 PROClisten(6)
1080 PROCatn_false
 .
 .
2000 REPEAT:UNTIL FNservice_requested
2010 statusbyte% = FNserial_poll(7)
 .
 .
```

To return error status

```
1000 REM computer configured as system controller
1010 REM data transfer
1020 REM return error status
1030
1040 status% = FNXinit
1045 IF status% <> 0 THEN 10000
1050 status% = FNXunlisten
1055 IF status% <> 0 THEN 10000
1060 status% = FNXtalk(7)
1065 IF status% <> 0 THEN 10000
1070 status% = FNXlisten(6)
1075 IF status% <> 0 THEN 10000
1080 status% = FNXatn_false
1085 IF status% <> 0 THEN 10000
 .
 .
2000 REPEAT
2002 status% = FNXservice_requested(state%)
2005 IF status% <> 0 THEN 10000
2007 UNTIL state%
2010 status% = FNXserial_poll(statusbyte%,7)
2015 IF status% <> 0 THEN 10000
 .
 .
```

Line 1040 initialises the interface hardware, sends REN (remote enable) true and IFC (interface clear) true for approximately 650 microseconds. The Computer becomes controller in charge

Line 1050 unaddresses any active listeners

Line 1060 addresses device 7 to talk. If another device is addressed to talk it is unaddressed. If the device has an extended address PROCtalkext should be used

Line 1070 addresses device 6 to listen. If the device has an extended address PROClistenext should be used

Line 1080 sends ATN (attention) false. The Computer relinquishes active control of the bus and the data transfer commences

Often the talker will request service from the controller in charge and indicate in its status byte that the output of data has finished.

Line 2000 (or 2000-2007) determines whether a device or devices are requesting service by sending SRQ (service request)

Line 2010 conducts a serial poll of the device with address 7 (the talker)

4.5 Determining the Status of a Device

4.5.1 Serial Poll

If a device requires action from the Computer, for example it has completed a measurement, it can request service from the Computer by sending SRQ (service request). To determine whether a device or devices are requesting service

```
state% = FNservice_requested
```

or

```
status% = FNXservice_requested(state%)
```

returns the logical state TRUE in the variable state% if a device or devices are requesting service.

A serial poll is conducted to determine which device or devices are requesting service. Each device in a system with a serial poll capability must be polled in turn. When a device is polled it returns a single STB (status byte). If bit 6 is set then the device is sending RQS (request service). The manufacturer' s manual for a device must be consulted to determine the meaning of the other 7 bits of the status byte. To conduct a serial poll of the device with address 10

```
statusbyte% = FNserial_poll(10)
```

or

```
status% = FNXserial_poll(statusbyte%,10)
```

returns the STB (status byte) in the variable statusbyte%.

During a serial poll each device must be polled separately. Therefore, a serial poll is time-consuming.

4.5.2 Parallel Poll

A parallel poll is a rapid way of determining whether a device or devices require action by the Computer. Before a parallel poll is conducted those devices that are to take part must be assigned a bit in the parallel poll response byte. This is done either remotely or locally. When a device is remotely configured to enable it to take part in a parallel poll, the least significant four bits of the PPE (parallel poll enable) byte (the S, P3, P2 and P1 bits) specify the sense and assigned bit in the PPR (parallel poll response) byte as shown by the table in Appendix III. To remotely configure a device enable it to take part in a parallel poll

Example
To abort on error

```
2500 PROCparallel_poll_unconfigure
2510 PROCparallel_poll_enable(4,12)
```

or to return error status

```
2500 status% = FNXparallel_poll_unconfigure
2505 IF status% <> 0 THEN 10000
2510 status% = FNXparallel_poll_enable(4,12)
2515 IF status% <> 0 THEN 10000
  .
  .
```

Line 2500 ensures that all devices with a parallel poll capability are unconfigured and are, therefore, unable to take part in a subsequent parallel poll until enabled.

Line 2510 enables the device with address 4 to take part in a parallel poll. The table in Appendix III shows that the S (sense) bit 3 is set and PPR5 (parallel poll response) is specified by the integer ' 12' .

During a parallel poll each enabled device compares its internal individual status bit with the sense bit it received when it was enabled. If they are the same, the bit assigned to that device is set in the parallel poll response byte.

To conduct a parallel poll

```
response% = FNparallel_poll
```

or

```
status% = FNXparallel_poll(response%)
```

returns the PPR (parallel poll response) byte in the variable response%.

To remotely disable the device with address 4 from taking part in a parallel poll

```
PROCparallel_poll_disable(4)
```

or

```
status% = FNXparallel_poll_disable(4)
```

If a data transfer which does not involve the Computer is taking place when a parallel poll is conducted data loss or corruption can occur as the Computer sets ATN (attention) true asynchronously. To prevent this ATN (attention) must be set true synchronously before the parallel poll. To resume data transfer after the parallel poll ATN (attention) must be set false.

```
PROCatn_true_sync
response% = FNparallel_poll
PROCatn_false
```

or

```
status% = FNXatn_true_sync
IF status% <> 0 THEN 10000
status% = FNXparallel_poll
IF status% <> 0 THEN 10000
status% = FNXatn_false
IF status% <> 0 THEN 10000
```

4.6 Initialising Devices

A device may have been programmed incorrectly or an error may have occurred. To clear a device with the capability

```
PROCselected_device_clear(6)
```

or

```
status% = FNXselected_device_clear(6)
```

initialises the device with address 6.

To clear all devices with the capability

```
PROCdevice_clear
```

or

```
status% = FNXdevice_clear
```

Note that device clear only affects device functions. It normally returns a device to its power on state. Interface functions are cleared by IFC (interface clear).

4.7 Triggering Devices

To trigger the basic operation of a device with the capability

```
PROCgroup_execute_trigger(14)
```

or

```
status% = FNXgroup_execute_trigger(14)
```

triggers the device with address 14.

4.8 Remote and Local Operation of Devices

Following initialisation the Computer as system controller sends REN (remote enable) true. Subsequent addressing of a device to listen changes it from local (front panel) to remote (interface) control. To return the device with address 9 to local control

```
PROCgo_to_local(9)
```

or

```
status% = FNXgo_to_local(9)
```

Use of the front panel switches also returns a device to local control. To ensure that all devices with the capability only respond to remote (interface) control

```
PROClocal_lockout
```

or

```
status% = FNXlocal_lockout
```

When a device is in the local lockout state it can be returned to local (front panel) control by

```
PROCgo_to_local(9)
```

or

```
status% = FNXgo_to_local(9)
```

To return all devices to local control and clear the local lockout state

```
PROCren_false
```

or

```
status% = FNXren_false
```

To enable remote operation of devices

```
PROCren_true
```

or

```
status% = FNXren_true
```

Subsequent addressing of a device to listen will again change it from local to remote operation.

4.9 Passing Control to Another Device

Only one system controller is permitted in an IEEE488 system and is the only device which can send REN (remote enable) and IFC (interface clear). Following initialisation, if the Computer is system controller, it sends REN (remote enable) and then IFC (interface clear) for approximately 650 microseconds and becomes controller in charge. It can then manage the system.

To determine whether the Computer is system controller

```
state% = FNsystem_controller
```

or

```
status% = FNXsystem_controller(state%)
```

returns the logical state TRUE in the variable state% if the Computer is system controller.

To pass control to a device with address 4 which has the capability to become controller in charge

```
PROCpass_control(4)
```

or

```
status% = FNXpass_control(4)
```

The Computer then acts as a talker/listener. It can become controller in charge again if either

the current controller in charge passes control back by addressing the Computer to talk and sending TCT (take control). This process is transparent to the user and can occur at any time.

or

if the Computer is the system controller, it sends IFC (interface clear).

To determine whether the Computer is controller in charge

```
state% = FNcontroller_in_charge
```

or

```
status% = FNXcontroller_in_charge(state%)
```

returns the logical state TRUE in the variable state% if the Computer is controller in charge.

5 The Computer as a Simple Talker/Listener

The Computer acts as a talker/listener if

Switch 6 of the DIP switch on the Expansion Card is in the on position ie it is not the system controller

or

as controller in charge it has passed control to another device.

To determine whether the Computer is a talker/listener

```
state% = FNcontroller_in_charge
```

or

```
status% = FNXcontroller_in_charge(state%)
```

returns the logical state FALSE in the variable state% if the Computer is a talker/listener.

The system controller normally sends IFC (interface clear) to return the interfaces of all the devices on the bus to a known initial state. To determine whether the Computer has received IFC (interface clear)

```
state% = FNinterface_cleared
```

or

```
status% = FNXinterface_cleared(state%)
```

returns the logical state TRUE in the variable state% if the Computer has received IFC (interface clear).

5.1 Data Input and Output - strings of up to 255 characters

Example
To abort on error

```
1000 REM computer configured as simple talker/listener
1010 REM data input and output
1020 REM strings of up to 255 characters
1030 REM abort on error
1040
1050 PROCeos_in_1chr(10)
1060 PROCeos_out_1chr(10)
1070 REPEAT:UNTIL FNinterface_cleared
1080 message$ = FNinput(0,255)
1090 PRINT "Message received ";message$
1100 PROCoutput(0,"ACORN ARCHIMEDES")
 .
 .
```

to return error status

```
1000 REM computer configured as simple talker/listener
1010 REM data input and output
1020 REM strings of up to 255 characters
1030 REM return error status
1040
1050 status% = FNXeos_in_1chr(10)
```

```
1055 IF status% <> 0 THEN 10000
1060 status% = FNXeos_out_1chr(10)
1065 IF status% <> 0 THEN 10000
1070 REPEAT
1072 status% = FNXinterface_cleared(state%)
1075 IF status% <> 0 THEN 10000
1077 UNTIL state% = TRUE
1080 status% = FNXinput(message$,0,255)
1085 IF status% <> 0 THEN 10000
1090 PRINT "Message received ";message$
1100 status% = FNXoutput(0,"ACORN ARCHIMEDES")
1105 IF status% <> 0 THEN 10000
 .
 .
```

Line 1050 specifies the end of string character for input as LF (line feed, ASCII code 10)

Line 1060 specifies the end of string character for output as LF (line feed, ASCII code 10)

Line 1070 (or 1070-1077) delays the program until IFC (interface clear) has been received from the system controller

Line 1080 inputs up to 255 characters, when the Computer is addressed to listen by the controller in charge, in mode 0 ie until the end of string character specified on line 1050 is received, and returns it in the variable message$

Line 1100 outputs the string ACORN ARCHIMEDES, when the Computer is addressed to talk by the current controller in charge, in mode 0 ie the end of string character specified on line 1060 is appended.

5.2 Data Input and Output - directly to and from memory

Example
To abort on error

```
1000 REM computer configured as simple talker/listener
1010 REM data input and output
1020 REM binary coded data directly to and from memory
1030 REM abort on error
1040
1050 DIM buffer% 2047
1060 REPEAT:UNTIL FNinterface_cleared
1070 PROCinputmem(1,buffer%,2048)
1080
1090 REM save data in file on disc
1100
1110 OSCLI "SAVE data "+STR$~buffer%+" + 800"
1120
 .
 .
3000 REM load data from file on disc
3010
3020 OSCLI "LOAD data "+STR$~buffer%
3030
3040 PROCoutputmem(1,buffer%,2048)
 .
 .
```

To return error status

```
1000 REM computer configured as simple talker/listener
1010 REM data input and output
```

```
1020 REM binary coded data directly to and from memory
1030 REM return error status
1040
1050 DIM buffer% 2047
1060 REPEAT
1062 status% = FNXinterface_cleared(state%)
1065 IF status% <> 0 THEN 10000
1067 UNTIL state%
1070 status% = FNXinputmem(1,buffer%,2048)
1075 IF status% <> 0 THEN 10000
1080
1090 REM save data in file on disc
1100
1110 OSCLI "SAVE data "+STR$~buffer%+" + 800"
1120
 .
 .
3000 REM load data from file on disc
3010
3020 OSCLI "LOAD data "+STR$~buffer%
3030
3040 status% = FNXoutputmem(1,buffer%,2048)
3045 IF status% <> 0 THEN 10000
 .
 .
```

Line 1050 reserves 2048 bytes of memory for the data

Line 1060 (or 1060-1067) delays the program until IFC (interface clear) has been received from the system controller

Line 1070 inputs up to 2048 bytes of data, when the Computer is addressed to listen by the current controller in charge, in mode 1 ie until EOI (end or identify) true is received with a data byte and stores it in memory starting at address buffer%

Line 3040 outputs 2048 bytes of data from memory starting at address buffer%, when the Computer is addressed to talk by the current controller in charge, in mode 1 ie EOI (end or identify) is set true with the last data byte

5.3 Requesting Service from the Controller in Charge

If the Computer requires action from the controller in charge, for example it has completed a series of calculations and has the results ready for output, it can request service from the controller in charge by sending SRQ (service request). The controller in charge conducts a serial poll to determine which device or devices are requesting service. Each device in a system with a serial poll capability is polled in turn. When the Computer is polled it returns a single STB (status byte). If the Computer is requesting service bit 6, the RQS (request service) bit, is set. The user must specify the meaning of the other seven bits of the status byte. To request service from the controller in charge

```
PROCrequest_service(3)
```

or

```
status% = FNXrequest_service(3)
```

sets the least two significant bits of the STB (status byte).

Following the reading of the STB (status byte) by the controller in charge during a serial poll, the RQS (request service) bit is cleared and SRQ (service request) is sent false automatically.

6 BASIC Procedures and Functions

The text file, Docs.Chap6, on the distribution disc, describes in alphabetical order the BASIC procedures and functions under the following headings.

Purpose
A description of what the procedure or function does.

Syntax
A formal declaration of both the abort on error form and the return error status form of the procedure or function.

Parameters
A list of the parameters required, giving their type, range and, if necessary, additional information on their specification.

Results
A list of the results returned, giving their type, range and, if necessary, additional information.

Example
Example to illustrate the use of both the abort on error form and the return error status form of the procedure or function. For clarity the parameters of procedures are shown as constants. However, variables of the same type can be used.

Notes
Give details of the operation of the interface software and when a procedure or function has the same effect as a number of lower level procedures or functions these are listed.

Remote messages
Shows the logical state of the ATN (attention) uniline message together with the multiline messages sent and received. LAD represents a listen address of a specific device. TAD represents a talk address of a specific device.

7 Using the IEEE488 Interface from Assembler

Read the Archimedes Programmer' s Reference Manual, particularly the sections on Fundamental Operating System Concepts and Appendix A on ARM Assembler, before writing programs in assembler. General information on communicating with the operating system is given in the Programmer' s Reference Manual.

The IEEE488 interface software is a relocatable module which extends the operating system. The facilities of the IEEE488 interface are accessed through a single Intelligent Interfaces IEEE488 SWI (software interrupt) number &40340 (error generating) or &60340 (error returning) allocated by Acorn Computers. If an error occurs in an error generating SWI, on return the V flag is set and register R0 contains a pointer. The pointer is to an error block. The format of the error block is

first four bytes - a word giving the error number

subsequent bytes -a string terminated by a zero byte giving the error message

The error number is formed by combining the IEEE488 error identifier &00800600 allocated by Acorn Computers and an IEEE488 interface software basic error number. Appendix IV lists the IEEE488 errors.

Parameters are passed and results are returned in the processor' s registers. Appendix II gives the allocation of parameters and results to the processor' s registers. All commands share the same structure.

On entry
- R0 is undefined
- R1 is command number
- R2-R7 parameters

On exit
- R0 points to an error block
- R1 is preserved
- R2-R7 parameters are preserved, results are updated

Example
To abort on error

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM abort on error
1030
1040 DIM code% 256
1050 IEEE488% = &40340
1060 specwreos% = &03
1070 specrdeos% = &04
1080 retryon% = &06
1085 init% = &0A
1087 sre% = &0E
1090 ifc% = &0D
1100 specwrmode% = &08
1110 specrdmode% = &09
1120 adrtolistenwrdataunlgts% = &3C
1130 adrtotalkrddatauntgts% = &3D
1140
1150 FOR pass% = 0 TO 2 STEP 2
1160 P% = code%
1170 [OPT  pass%
1180 MOV R1,#specwreos%
1190 MOV R2,#1
```

```
1200 MOV R3,#10
1210 SWI IEEE488%
1220 MOV R1,#specrdeos%
1230 SWI IEEE488%
1240 MOV R1,#retryon%
1250 SWI IEEE488%
1252 MOV R1,#init%
1254 SWI IEEE488%
1256 MOV R1,#sre%
1258 SWI IEEE488%
1260 MOV R1,#ifc%
1270 SWI IEEE488%
1280 MOV R1,#specwrmode%
1290 MOV R2,#0
1300 SWI IEEE488%
1310 MOV R1,#adrtolistenwrdataunlgts%
1320 MOV R3,#16
1330 ADR R5,beginoutputdata
1340 MOV R6,#(endoutputdata - beginoutputdata)
1350 SWI IEEE488%
1360 MOV R1,#specrdmode%
1370 MOV R2,#0
1380 SWI IEEE488%
1390 MOV R1,#adrtotalkrddatauntgts%
1400 MOV R3,#16
1410 ADR R5,begininputdata
1420 MOV R6,#(endinputdata - begininputdata)
1430 SWI IEEE488%
1440 MOVS R15,R14
1450
1460 .beginoutputdata
1470 EQUS "F0T1R3X"
1480 .endoutputdata
1490
1500 .begininputdata
1510 EQUS STRING$(9," ")
1520 .endinputdata
1530 ]
1540 NEXT pass%
1550 CALL  code%
 .
 .
```

or to return error status

```
1000 REM computer configured as system controller
1010 REM data output and input
1020 REM to return error status
1030
1040 DIM code% 256
1050 XIEEE488% = &60340
1055 OS_WriteO% = &02
1060 specwreos% = &03
1070 specrdeos% = &04
1080 retryon% = &06
1085 init% = &0A
1087 sre% = &0E
1090 ifc% = &0D
1100 specwrmode% = &08
1110 specrdmode% = &09
1120 adrtolistenwrdataunlgts% = &3C
```

```
1130 adrtotalkrddatauntgts% = &3D
1140
1150 FOR pass% = 0 TO 2 STEP 2
1160 P% = code%
1170 [OPT  pass%
1180 MOV R1,#specwreos%
1190 MOV R2,#1
1200 MOV R3,#10
1210 SWI XIEEE488%
1215 BVS error
1220 MOV R1,#specrdeos%
1230 SWI XIEEE488%
1235 BVS error
1240 MOV R1,#retryon%
1250 SWI XIEEE488%
1251 BVS error
1252 MOV R1,#init%
1254 SWI XIEEE488%
1255 BVS error
1256 MOV R1,#sre%
1258 SWI XIEEE488%
1259 BVS error
1260 MOV R1,#ifc%
1270 SWI XIEEE488%
1275 BVS error
1280 MOV R1,#specwrmode%
1290 MOV R2,#0
1300 SWI XIEEE488%
1305 BVS error
1310 MOV R1,#adrtolistenwrdataunlgts%
1320 MOV R3,#16
1330 ADR R5,beginoutputdata
1340 MOV R6,#(endoutputdata - beginoutputdata)
1350 SWI XIEEE488%
1355 BVS error
1360 MOV R1,#specrdmode%
1370 MOV R2,#0
1380 SWI XIEEE488%
1385 BVS error
1390 MOV R1,#adrtotalkrddatauntgts%
1400 MOV R3,#16
1410 ADR R5,begininputdata
1420 MOV R6,#(endinputdata - begininputdata)
1430 SWI XIEEE488%
1435 BVS error
1440 MOVS R15,R14
1450
1451 .error
1452 ADR R0,errormessage
1453 SWI OS_WriteO%
1454 MOVS R15,R14
1455
1456 .errormessage
1457 EQUS "IEEE488 error " + CHR$(10) + CHR$(13)
1458 EQUB &00
1459
1460 .beginoutputdata
1470 EQUS "F0T1R3X"
1480 .endoutputdata
1490
1500 .begininputdata
```

```
1510 EQUS STRING$(9," ")
1520 .endinputdata
1530 ]
1540 NEXT pass%
1550 CALL  code%
 .
 .
```

Line 1050 defines the IEEE488 SWI

Lines 1060-1130 defines the IEEE488 command numbers

Lines 1180-1230 specify the end of string character for output and input as LF (line feed, ASCII code 10)

Lines 1240-1250 enable timeouts

Lines 1252-1270 initialises the interface hardware, sends REN (remote enable) and IFC (interface clear) true for approximately 650 microseconds. The Computer becomes controller in charge

Lines 1280-1350 addresses device 16 to listen and outputs the string F0T1R3X from memory beginning at beginoutputdata in mode 0 ie the end of string character specified by lines 1180-1230 is appended

Lines 1360-1430 addresses device 16 to talk and inputs up to 9 characters in mode 0 ie until the end of string character specified by lines 1180-1230 is received and places them in memory beginning at begininputdata

If an error occurs both of the above examples produce a similar result ie an error message is displayed. However, in practice the return error status SWI can be used to trap errors in the program and take the appropriate corrective action.

8 Hardware

The interface uses the Texas Instruments TMS9914A GPIB Adapter, together with the SN75160A and SN75162A octal GPIB transceivers.

The TMS9914A interfaces the Computer to the IEEE488 bus. It relieves the processor of the task of handling IEEE488 protocol when commands and data are sent and received and implements most of the IEEE488 functions. If processor intervention is required in order to support a function the TMS9914A generates an interrupt. The IEEE488 interface software utilises these features to, for example, allow the Computer to be remotely configured to enable it to take part in a parallel poll in a way that is transparent to the user. A 5 MHz crystal provides the TMS9914A with a clock for timing purposes at the maximum permitted frequency. This results in the highest possible data transfer rates.

The SN75160A uses tri-state drivers to drive the data lines of the bus except during a parallel poll when it automatically switches to open collector drivers.

The Expansion Card has a 6-way DIP Switch. Switches 1-5 specify the IEEE488 Interface Address of the Computer. Switch 6 specifies whether the Computer is either the system controller (off) or a simple talker/listener (on). Appendix I lists the DIP switch settings.

The interface software is stored in a paged ROM and is automatically loaded and initialised by the operating system when the Computer is switched on or reset. The three option selection links (S1, S2 and S3) are provided to enable different sizes of EPROM to be fitted.

Size Kbyte	S1	S2	S3
8	A	A	N/A
16	A	A	N/A
32	B	A	N/A
64	B	B	A
128	B	B	B

References

Ref 1

ANSI/IEEE Std 488-1978
(Revision of ANSI/IEEE Std 488-1975)
(includes supplement IEEE Std 488A-1980)
IEEE488 Standard Digital Interface for Programmable Instrumentation
The Institute of Electrical and Electronic Engineers Inc 1978

Ref 2

TMS 9914 GPIB Adapter Data Manual
Texas Instruments

Appendix I

EXPANSION CARD DIP SWITCH SETTINGS

Computer Configured as the System Controller

Address	SW1	SW2	SW3	SW4	SW5	SW6
0	OFF	OFF	OFF	OFF	OFF	OFF
1	ON	OFF	OFF	OFF	OFF	OFF
2	OFF	ON	OFF	OFF	OFF	OFF
3	ON	ON	OFF	OFF	OFF	OFF
4	OFF	OFF	ON	OFF	OFF	OFF
5	ON	OFF	ON	OFF	OFF	OFF
6	OFF	ON	ON	OFF	OFF	OFF
7	ON	ON	ON	OFF	OFF	OFF
9	ON	OFF	OFF	ON	OFF	OFF
10	OFF	ON	OFF	ON	OFF	OFF
11	ON	ON	OFF	ON	OFF	OFF
12	OFF	OFF	ON	ON	OFF	OFF
13	ON	OFF	ON	ON	OFF	OFF
14	OFF	ON	ON	ON	OFF	OFF
15	ON	ON	ON	ON	OFF	OFF
16	OFF	OFF	OFF	OFF	ON	OFF
17	ON	OFF	OFF	OFF	ON	OFF
18	OFF	ON	OFF	OFF	ON	OFF
19	ON	ON	OFF	OFF	ON	OFF
20	OFF	OFF	ON	OFF	ON	OFF
21	ON	OFF	ON	OFF	ON	OFF
22	OFF	ON	ON	OFF	ON	OFF
23	ON	ON	ON	OFF	ON	OFF
24	OFF	OFF	OFF	ON	ON	OFF
25	ON	OFF	OFF	ON	ON	OFF
26	OFF	ON	OFF	ON	ON	OFF
27	ON	ON	OFF	ON	ON	OFF
28	OFF	OFF	ON	ON	ON	OFF
29	ON	OFF	ON	ON	ON	OFF
30	OFF	ON	ON	ON	ON	OFF

Note - address 31 is not permitted

Computer Configured as a Simple Talker/Listener

Address	SW1	SW2	SW3	SW4	SW5	SW6
0	OFF	OFF	OFF	OFF	OFF	ON
1	ON	OFF	OFF	OFF	OFF	ON
2	OFF	ON	OFF	OFF	OFF	ON
3	ON	ON	OFF	OFF	OFF	ON
4	OFF	OFF	ON	OFF	OFF	ON
5	ON	OFF	ON	OFF	OFF	ON
6	OFF	ON	ON	OFF	OFF	ON
7	ON	ON	ON	OFF	OFF	ON
9	ON	OFF	OFF	ON	OFF	ON
10	OFF	ON	OFF	ON	OFF	ON
11	ON	ON	OFF	ON	OFF	ON
12	OFF	OFF	ON	ON	OFF	ON
13	ON	OFF	ON	ON	OFF	ON
14	OFF	ON	ON	ON	OFF	ON
15	ON	ON	ON	ON	OFF	ON
16	OFF	OFF	OFF	OFF	ON	ON
17	ON	OFF	OFF	OFF	ON	ON
18	OFF	ON	OFF	OFF	ON	ON
19	ON	ON	OFF	OFF	ON	ON

20	OFF	OFF	ON	OFF	ON	ON
21	ON	OFF	ON	OFF	ON	ON
22	OFF	ON	ON	OFF	ON	ON
23	ON	ON	ON	OFF	ON	ON
24	OFF	OFF	OFF	ON	ON	ON
25	ON	OFF	OFF	ON	ON	ON
26	OFF	ON	OFF	ON	ON	ON
27	ON	ON	OFF	ON	ON	ON
28	OFF	OFF	ON	ON	ON	ON
29	ON	OFF	ON	ON	ON	ON
30	OFF	ON	ON	ON	ON	ON

Note - address 31 is not permitted